Verträge und objektorientierter Entwurf

Überblick

Was dieses Video behandelt:

- **Design by Contract** (etwa: Entwurf gemäß Vertrag) als Richtlinie beim objektorientierten Entwurf
  - Verträge
  - Vererbung
  - Invarianten
- Überprüfung von Argumenten und Verträgen
  - `IllegalArgumentException`, `NullPointerException`
  - Assertions

Modularität

Aufteilung des Gesamtprogramm in getrennte Komponenten:

- Jede Komponente hat eine klar spezifizierte Funktion.
- Gesamtprogramm wird so zusammengesetzt, dass es korrekt ist, wenn nur alle Komponenten ihre Spezifikation erfüllen.

Viele Gründe, z.B.:

- getrennte und parallele Entwicklung
- Austauschbarkeit
- Wiederverwendbarkeit
- Abstraktion von Implementierungsdetails

Design by Contract

Design by Contract enthält methodologische Richtlinien zur komponentenbasierten Softwareentwicklung.

Analogie zu Verträgen im Geschäftsleben:

- Jede Komponente bietet einen Vertrag an.
- Ein Vertrag beinhaltet das Versprechen einer Programmkomponente, eine bestimmte Leistung zu erbringen, wenn bestimmte Voraussetzungen erfüllt sind.
- Zusammensetzung von Komponenten nur nach Vertrag:
  Das Gesamtprogramm ist korrekt, wenn nur alle Komponenten ihre Verträge einhalten.

Vertrag einer Klasse

Der Vertrag wird in der Dokumentation angegeben und enthält:

- eine Vertragsklausel für jede Methode
- Klasseninvarianten:
  Die Klasse garantiert, dass ihre Objekte stets eine bestimmte Eigenschaft haben.

Vertrag einer Methode

Der Vertrag einer Methode besteht aus

Vorbedingung, **Nachbedingung** und **Effektbeschreibung**.

Sinngemäß:

Wenn beim Aufruf der Methode die Vorbedingung X erfüllt ist, dann wird die Leistung Y erbracht. Die Ausführung der Methode hat den Nebeneffekt Z (z.B. I/O).

- Die **Vorbedingung** bezieht sich auf die Argumente der Methode und die Werte der Instanzvariablen vor Aufruf der Methode.
- Die **Nachbedingung** bezieht sich auf das Ergebnis der Methode und die neuen Werte der Instanzvariablen.
- Die Methode verspricht, die Nachbedingung herzustellen, wenn der Aufrufer die Vorbedingung sichergestellt hat.

Beispiel

Interface `World` aus Game of Life:

```
/**
 * Setzt die Lebendigkeitseigenschaft der Zelle
 * mit den Koordinaten (x, y).
 *
 * @param x x-Koordinate der Zelle
 * @param y y-Koordinate der Zelle
 * @param alive
 *    ob die Zelle nach Ausführung lebendig sein soll oder nicht
 * @throws IllegalArgumentException
 *    falls 0 <= x < getWidth()
 *    oder 0 <= y < getHeight() nicht gilt
 */
public void setCell(int x, int y, boolean alive);
```

Beispiel

Interface `World` aus Game of Life:

```
/**
 * Setzt die Lebendigkeitseigenschaft der Zelle
 * mit den Koordinaten (x, y).
 *
 * @param x x-Koordinate
 * @param y y-Koordinate
 * @param alive
 *    ob die Zelle nach Ausführung lebendig sein soll oder nicht
 * @throws IllegalArgumentException
 *    falls 0 <= x < getWidth()
 *    oder 0 <= y < getHeight() nicht gilt
 */
public void setCell(int x, int y, boolean alive);
```

Vorbedingung:

```
0 <= x  &&  x < getWidth() &&
0 <= y  &&  y < getHeight()
```

Beispiel

Interface `World` aus Game of Life:

```
/**
 * Setzt die Lebendigkeitseigenschaft der Zelle
 * mit den Koordinaten (x, y).
 *
 * @param x x-Koordinate
 * @param y y-Koordinate
 * @param alive
 *    ob die Zelle nach Ausführung lebendig sein soll oder nicht
 * @throws IllegalArgumentException
 *    falls 0 <= x < getWidth()
 *    oder 0 <= y < getHeight()
 */
public void setCell(int x, int y, boolean alive);
```

Vorbedingung:

```
0 <= x  &&  x < getWidth() &&
0 <= y  &&  y < getHeight()
```

Nachbedingung:

```
alive == isCellAlive(x, y)
```

Beispiel

Klasse `java.lang.Object`

- `equals`
- `hashCode`

Verträge und Vererbung

Eine Unterklasse ist an den Vertrag der Oberklasse gebunden.

- Eine überschreibende Methode muss den Vertrag der überschriebenen Methode einhalten:
  - Vorbedingungen können höchstens abgeschwächt werden.
  - Nachbedingungen können höchstens verstärkt werden.

Entwicklungsprinzip

- Benutze nur die im Vertrag zugesichterten Leistungen.
- Stelle Vorbedingungen stets sicher.

Getrennte Entwicklung von (gegenseitig abhängigen) Klassen:

- Spezifiziere die Schnittstellen und Verträge der Klassen.
- Jede Klasse kann nun nur anhand der Verträge der anderen Klassen implementiert werden.

Invarianten

- Oft sollen Instanzvariablen zu jedem Zeitpunkt bestimmte Eigenschaften haben. Beispiel: Instanzvariable `!= null`
- Anstatt sie in allen Vor- und Nachbedingungen aufzuführen formuliert man sie als **Invariante** der Klasse.
- Eine Invariante wird nur einmal für die Klasse formuliert, muss jedoch in allen Methoden und Konstruktoren beachtet werden.
  - Jeder Konstruktor muss die Invariante als Nachbedingung herstellen.
  - Jede Methode darf die Invariante als Vorbedingung annehmen.
  - Jede Methode muss die Invariante als Nachbedingung sicherstellen.

Invarianten

Beispiele:

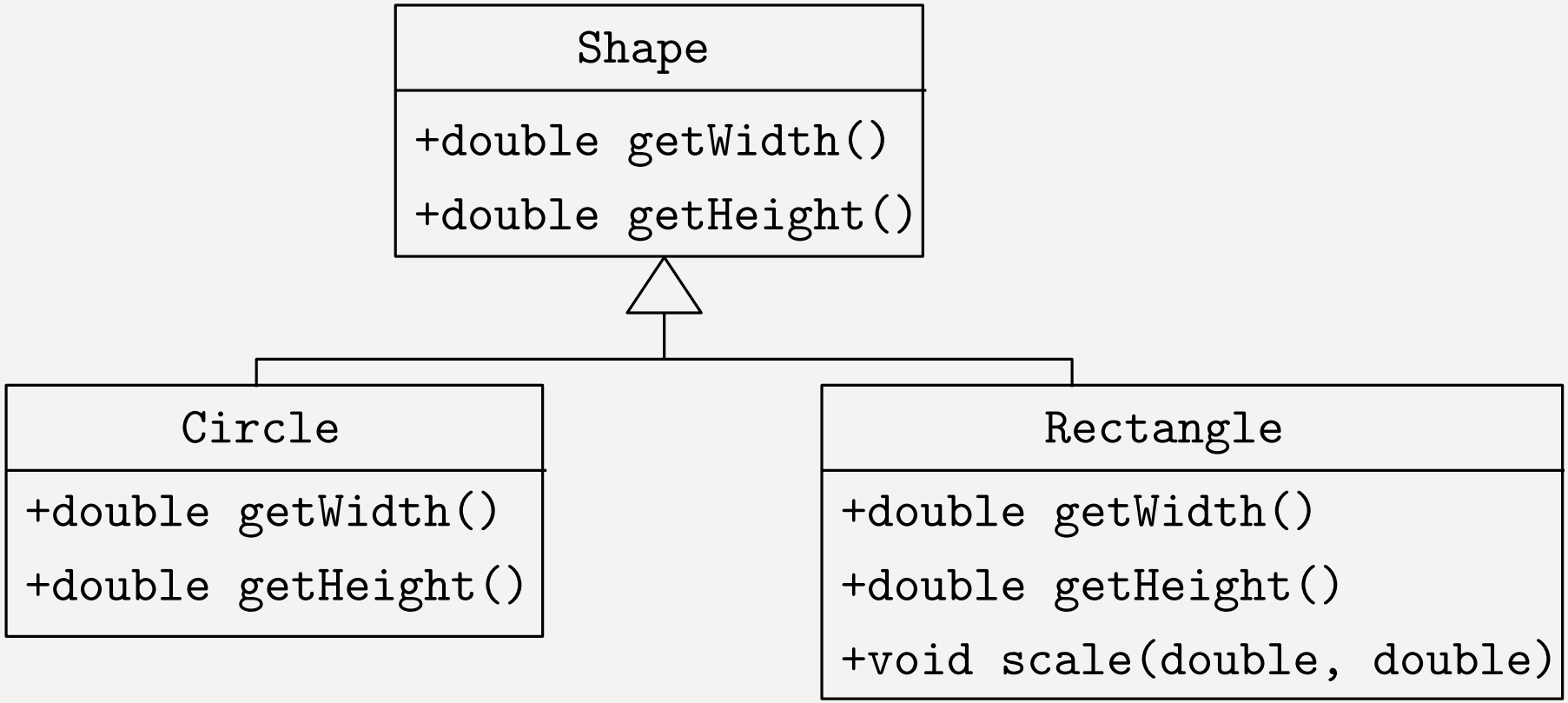
- Instanzvariable `!= null`
- redundante Datenhaltung

Verträge und Vererbung

- Eine Unterklasse ist an den Vertrag der Oberklasse gebunden.
- Jede überschriebene Methode muss den Vertrag der überschriebenen Methode einhalten:
  - Vorbedingungen können höchstens abgeschwächt werden.
  - Nachbedingungen können höchstens verstärkt werden.
- In der Unterklasse müssen alle Invarianten der Oberklasse erhalten bleiben.
  - Konstruktoren der Unterklasse müssen die Invarianten herstellen.
  - Überschriebene Methoden müssen die Invariante erhalten.

Verträge und Entwurf

Beispiel: Wir wollen folgende Klassenhierarchie um eine Klasse `Square` für Quadrate erweitern.



Sollen wir `Square` als Unterklasse von `Rectangle` modellieren?

Verträge und Entwurf

Vertrag von `void scale(double sx, double sy)`:

- Vorbedingung: `sx` und `sy` beide nichtnegativ.
- Nachbedingung: Es gilt `getWidth( ) == sx * w`, wobei `w` das Ergebnis von `getWidth()` vor dem Aufruf von `scale` ist.
  (und analog für die Höhe)

Die Klasse `Square` kann diesen Vertrag nicht erfüllen.

`Square` kann nicht als Unterklasse von `Rectangle` eingeordnet werden, sondern von `Shape`.

Überprüfung von Parametern und Verträgen

Öffentliche Methoden

Java-Konvention:
Jede öffentliche (`public`) Methode überprüft zuerst ihre Argumente und die Vorbedingung.

- `IllegalArgumentException` soll ausgelöst werden, wenn die Argumente die Vorbedingung nicht erfüllen.
- `NullPointerException` soll ausgelöst werden, wenn in den Argumenten ein `null`-Wert auftaucht, wo keiner erlaubt ist.
- ähnliche Spezialfälle, z.B. `IndexOutOfBoundsException`

N.B.: Bei privaten Methoden kann man (wenigstens theoretisch) ausschließen, dass illegale Argumente übergeben werden.

Assertions

Wir haben Verträge bisher nur als Teil der informellen Dokumentation betrachtet.

- In einem korrekten Programm halten sich alle Programmteile an die Verträge.
- Um Programmierfehler frühzeitig zu erkennen, ist es günstig die Einhaltung der Verträge im Programm selbst zu überprüfen.
- Im fertigen Programm können diese Tests dann wieder abgeschaltet werden.

Assertions

Java stellt eine Zusicherungsanweisung zur Überprüfung von Verträgen bereit:

```
assert test : errorValue;
```

- Verhält sich wie

  ```
  if (!test) { throw new AssertionError(errorValue); }
  ```

- Standardmäßig werden Assertions ignoriert, d.h. sie haben keinen Einfluss auf den Progammablauf oder die Geschwindigkeit.
- Assertion müssen erst aktiviert werden:
  `java -ea Main` (enable assertions)
  (In Eclipse `-ea` zu "VM arguments" hinzufügen).

Assertions

Zusicherungen werden zur Überprüfung von Eigenschaften verwendet, die nach Vertrag garantiert gelten müssen.

- Im Idealfall wird der Vertrag eingehalten, die assert-Instruktionen machen gar nichts.
- Ist der Vertrag verletzt, so führen die assert-Instruktionen zum Programmabbruch und zur Meldung des Vertragsbruchs.
- Assertions werden nur zum Testen des Vertrags benutzt. Das Verhalten des Programms soll vom Ein- oder Ausschalten von Assertions unbeeinflusst sein.

Assertions

Zusicherungen werden zur Überprüfung von Eigenschaften verwendet, die nach Vertrag garantiert gelten müssen.

- Nach Konvention werden die Vorbedingungen an Argumente in öffentlichen Methoden **nicht** mit `assert` geprüft. Dort soll z.B. `IllegalArgumentException` ausgelöst werden.
- Die Argumente and Vorbedingungen in privaten Methoden können jedoch mit `assert` überprüft werden.
- Invarianten und Nachbedinungen werden mit `assert` überprüft.

Assertions

Beispiele:

- Interne Invarianten, Kontrollflussinvarianten
- Vorbedingungen in privaten Methoden
- Nachbedingungen
- Überprüfung von Invarianten: `BitSet`

Zusammenfassung

- Design by Contract
- **Entwicklungsprinzipen**:
  - Benutze nur die im Vertrag einer Klasse zugesichterten Leistung, stelle Vorbedingungen stets sicher.
  - Dokumentiere Klasseninvarianten.
  - Minimiere das öffentliche Interface von Klassen.
- Verwende Assertions zur programmatischen Dokumentation und Überprüfung von Verträgen.